Introduction to Mobile Development- Lab 4

Objects in Dart Programming Language

By Ali Mujtaba- 100792180

A logo for a college

Description automatically generated

Ali Mujtaba- 100792180

INFT3101- Durham College

Professor Mohammed Shamas

October 30th, 2024

Contents

[What are Objects? 3](#_Toc181361032)

[1.1: How are Classes Relevant to Objects? 3](#_Toc181361033)

[1.2: Why Objects Are Used in Programming 3](#_Toc181361034)

[1.3: Comparison with Similar Concepts in Other Languages 3](#_Toc181361035)

[1.4: Advantages and Limitations of Using Objects in Dart 3](#_Toc181361036)

[1.5: Key Object Oriented Principles in Dart 4](#_Toc181361037)

[Example implementation 5](#_Toc181361038)

[Example 1: Creating an object using constructors 5](#_Toc181361039)

[Example 2: Implementing other object oriented principles 6](#_Toc181361040)

[Real-World Example 8](#_Toc181361041)

# What are Objects?

In Dart, objects are at the core of how data is represented and manipulated; everything in Dart, even basic data types like integers and strings, is an object. In object-oriented programming languages, objects are instances of a class. They contain data (or in programming terms, properties) and behaviors (also called methods). They allow us to create reusable structures that can represent both simple and complex entities.

## 1.1: How are Classes Relevant to Objects?

Classes lay out the framework for an object to be created. For more context, here are the definitions of both terms.

* **Class:** A blueprint or template for creating objects. It defines the properties (fields) and methods (functions) that an object will inherit.
* **Object:** An instance of a class. When you create objects from classes, you allocate memory for the data, allowing you to call the methods defined within the class.

## 1.2: Why Objects Are Used in Programming

Objects provide a powerful structure for creating reusable components in code. They allow developers to model real-world entities with properties and behaviors, making code more modular and maintainable. This structure is particularly useful in Dart, as it promotes organized, readable code even in complex applications. By creating objects, developers can encapsulate related data and functions within a single entity, reducing redundancy and making their code easier to understand and work with.

For example, in an app for managing tasks, a Task object could encapsulate properties like title, description, and deadline, along with methods to edit or mark the task as complete.

## 1.3: Comparison with Similar Concepts in Other Languages

Objects in Dart are similar to objects in many other object-oriented languages like Java, Python, and JavaScript, but with certain differences.

**Dart vs. Java:** In Java, everything is also object-oriented; however, Java requires more strict type declarations and interface implementations, while Dart is more flexible and dynamically typed.

**Dart vs. JavaScript:** JavaScript uses prototypes for inheritance, while Dart is more class-based. Dart’s class-based structure is more formalized and provides a clearer framework for defining and working with objects, while JavaScript’s prototypal inheritance allows more flexibility at the cost of formal structure.

Dart’s structure combines the organization of Java’s class system with some of the flexibility found in JavaScript, making it a good choice for both small and large projects.

## 1.4: Advantages and Limitations of Using Objects in Dart

**Advantages**

* **Encapsulation:** Dart allows data and methods to be grouped together in an object, making code more modular and secure.
* **Reusability:** Objects in Dart support reusability; developers can create classes once and instantiate multiple objects from them, reducing redundancy.
* **Polymorphism:** This feature allows for flexible code that can handle various object types through a shared parent class, making Dart code more adaptable.
* **Ease of Maintenance:** By grouping related properties and methods, objects make code easier to understand and maintain, especially in large applications.

**Limitations**

* **Complexity:** For small tasks, defining classes and creating objects can introduce unnecessary complexity.
* **Memory Overhead:** Dart’s object-oriented approach can be memory-intensive for performance-critical applications, as each object requires its own memory allocation.
* **Learning Curve:** New developers might find the concept of objects and classes more challenging than procedural code, as it introduces additional layers of abstraction.

## 1.5: Key Object Oriented Principles in Dart

1.5.1 Constructors

Constructors are special methods used to initialize objects, and Dart supports three types:

* **Default Constructor:** Automatically created if no constructor is defined.
* **Named Constructor:** Useful when you want multiple constructors with different names for clarity.
* **Factory Constructor:** Returns an existing instance instead of creating a new one, useful in cases like singleton patterns.

1.5.2 Properties and Methods

Throughout this report, there have been many mentions of properties and methods, so one may ask, what are they? Essentially, they are the data and behavior of an object.

* **Properties (Fields):** Variables defined in a class, representing the state or data of an object.
* **Methods (Functions):** Functions within a class that define the behavior of the object. These can manipulate object properties or execute tasks.

1.5.3 Encapsulation with Getters and Setters

In Dart, encapsulation is the practice of hiding an object’s internal state and only exposing specific parts. We achieve this using getters and setters, which are methods that control access to the object’s properties.

* **Getter:** A method used to retrieve a property’s value.
* **Setter:** A method used to set or update a property’s value, with optional validation.

1.5.4 Inheritance

Inheritance allows one class to inherit the properties and methods of another, making it a powerful tool for creating hierarchies and reusing code. In Dart, the extends keyword lets you create a subclass from an existing class, which means the subclass will have access to the parent class’s properties and methods.

1.5.5 Polymorphism

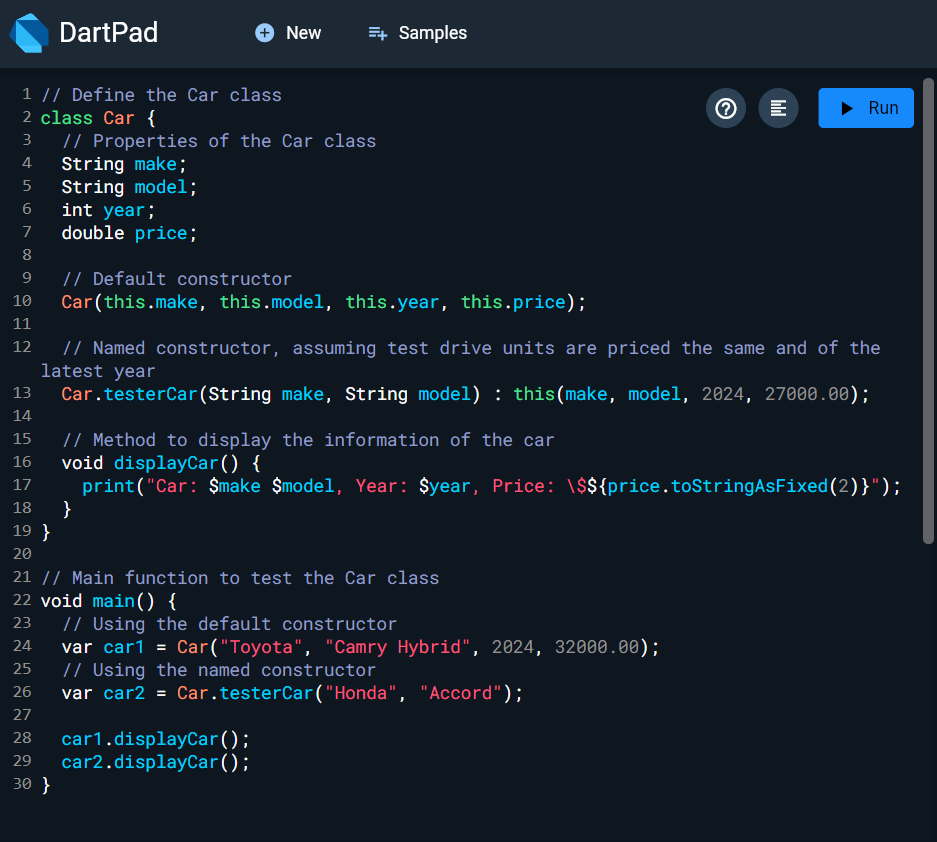
Polymorphism lets objects of different types be treated as instances of their parent class. This feature is especially useful when working with methods or lists that use a common type. For example, you can create a function that accepts an Animal type and pass in different subclasses of Animal.

# Example implementation

## Example 1: Creating an object using constructors

This example demonstrates a simple class called Car. Each Car represents an object with unique properties such as make, model, year and price. For this example, we will use both default and named constructors.

Source Code:



Output:

A screenshot of a computer

Description automatically generated

## Example 2: Implementing other object oriented principles

This example demonstrates a class hierarchy for grocery items. The base class, GroceryItem, includes properties like name, quantity, and price, with encapsulation to prevent negative values. The subclass PerishableItem inherits from GroceryItem and adds an expirationDate property. This structure allows us to maintain data integrity while enabling the creation of specialized item types.

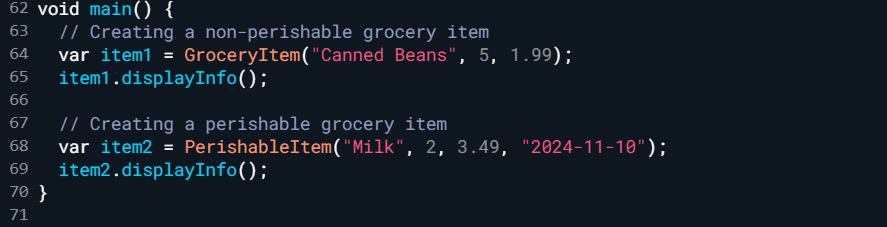
Source Code:

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated



Output:

A screen shot of a computer

Description automatically generated

# Real-World Example

Scenario: Shopping Cart in a Mobile App

In a Flutter mobile shopping app, objects can represent various entities within the app, like Product and CartItem. For instance, a Product class could define properties such as name, price, and stockQuantity, and methods like addToCart() and removeFromCart(). The CartItem class might encapsulate properties like the product (object reference), quantity, and a calculateTotal method to calculate the total cost.

By encapsulating these details in objects, developers could manage the shopping cart and products efficiently, ensuring that each item in the cart is an instance of CartItem, allowing easy addition, removal, and price calculation for items in the cart. This approach keeps the cart data separate and modular, letting the app easily interact with multiple CartItem objects, iterate through them, and update the UI whenever the cart’s state changes.

# References

Dart Language Tour. (n.d.). *Dart documentation*. Dart.dev. Retrieved October 30th, from https://dart.dev/language

Dart Programming – Classes and Objects. (n.d.). *W3Schools*. Retrieved October 30th, from <https://www.w3schools.io/languages/dart-class/>

DartPad. (n.d.). *Online Dart compiler*. Dart.dev. Retrieved October 31st, from <https://dartpad.dev/>

Oracle. (n.d.). *Object-Oriented Programming Concepts*. Oracle. Retrieved October 31st, from <https://docs.oracle.com/javase/tutorial/java/concepts/>